Implementation of Variable Precision Floating Point Fast Square Root Architecture

Pandrangi Aditya Sriram

EE3403 - Digital IC Design

December 5, 2024

References

- Advanced Components in the Variable Precision Floating Point Library: *Xiaojun Wang, Sherman Braganza, Miriam Lesser*
- Reciprocation, Square Root, Inverse Square Root, and Some Elementary Functions using Small Multipliers: *Milos Ercegovac, Jean-Michel Muller*

Floating-Point Square Root Architecture

- **Overview:** Uses small table lookup and takes advantage of Taylor expansion
- **Key Steps:**
  - **Reduction:** Reduces input size through initial lookup
  - **Evaluation:** Computes Taylor series terms for accuracy
  - **Post-Processing:** Multiplies result by corrective factor
- **Pipelining:** Linear/Non-Iterative, FPGA-friendly. Not implemented in the paper.
- **Utility:** Reduces hardware and time complexity by using only lookup tables and multipliers.

Input and Output

- 1 sign bit, n-Mantissa Bits and m-Exponent Bits.
- The current implementation is designed for IEEE Single Precision Floating Point Format ($n = 23$, $m = 8$). Can be generalized.
- Bottom-up approach for designing; Top-down approach for thinking.
- Let exponent $= E$, mantissa $= M$, and sign $= S$

Square Root Module

- Check for special cases. Else, compute in parallel:
- $S_{out} = 0$
- E_{out} =floor$[\frac{E_{in} - \text{Bias}}{2}]$+Bias, where Bias $= 2^{n-1} - 1$.
- M_{out} = FixedPointSquareRoot(M_{in}). If [E_{in} - Bias] is odd, subtract 1 and assign $M_{out} := \sqrt{2}M_{out}$
- Special Cases of Input:

Input	**Output**
$\pm$ Zero ($E_{in} = 0, M_{in} = 0$)	Zero (00000000)
Negative Non-zero ($S_{in} = 1$)	NaN ($7FC00000$)
+Inf ($S_{in} = 0, E_{in} = FF$ and $M_{in} = 0$)	+Inf ($7F800000$)
-Inf ($S_{in} = 1, E_{in} = FF$ and $M_{in} = 0$)	NaN ($7FC00000$)
NaN ($E_{in} = FF$ and $M_{in} \neq 0$)	NaN ($7FC00000$)

FixedPointSquareRoot Module

- **Input:** Y is an n-bit fixed point number corresponding to the fractional part (mantissa). Hidden bit assumed to be 1. Thus, the range is $[1, 2)$. Here,
- **Idea:** Let $k =$ceil$[\frac{n+1}{4}]$. Then, we write

$$\sqrt{Y} = \frac{\sqrt{Y\hat{R}}}{\sqrt{\hat{R}}} \tag{1}$$

- We want $\hat{R}$ such that $\sqrt{Y\hat{R}}$ is very close to 1, and thus Taylor Series can be used.
- $\hat{R}$ is a $(k+1)$-bit number whose square root is computed using a small LUT as $(k+1)$ is much smaller than n.
- Using an LUT, we calculate $\hat{R} = \frac{1}{Y^{(k)}}$, rounded down, where $Y^{(k)}$ is Y truncated to k bits.

Dealing with Numerator

- **Goal:** To calculate $\sqrt{Y\hat{R}}$
- As $Y\hat{R} \approx 1$, we define the error term, a $4k$-bit number

$$A = Y\hat{R} - 1 \tag{2}$$

- Now, as $Y^{(k)}$ is Y truncated to k^{th} bit, we have

$$Y^{(k)} \leq Y \leq Y^{(k)} + 2^{-k}$$
$$\implies 1 \leq \frac{Y}{Y^{(k)}} \leq 1 + 2^{-k} \quad \text{as } Y^{(k)} \geq 1$$

- Also, as $\frac{1}{Y^{(k)}} - 2^{-k-1} < \hat{R} \geq \frac{1}{Y^{(k)}}$,

$$\implies 1 - 2^{-k} < \hat{R}Y^{(k)} < 1$$
$$\implies \text{Combining,} 1 - 2^{-k} < \hat{R}Y^{(k)} < 1 + 2^{-k}$$
$$\implies -2^{-k} < A < +2^{-k}$$

Dealing with Numerator - Taylor Series

- Therefore, we can write

$$A = 0 + A_2 2^{-2k} + A_3 2^{-3k} + A_4 2^{-4k} \ldots \tag{3}$$

  where each $A_k \in \mathbb{Z}$, $|A_k| \leq 2^k - 1$
- Using Taylor Series, we get the first few terms as:

$$\sqrt{1+A} = 1 + \frac{A}{2} - \frac{1}{8} A_2^2 \cdot 2^{-4k} - \frac{1}{4} A_2 A_3 \cdot 2^{-5k} + \frac{1}{16} A_2^3 \cdot 2^{-6k} \tag{4}$$

- If we store A_2, A_3 as k-bit integers, their squares, products and cubes as n-bit integers, and the result $B = \sqrt{1+A}$ as n-bit fractional number (excluding hidden bit), then the required right shifts effectively are reduced by n:
  - Square: $4k - n$ (here, $28 - 23 = 5$)
  - Cube: $6k - n$ (here, $42 - 23 = 19$)
  - Product: $7k - n$ (here, $35 - 23 = 12$)

Negative value of A

- If A is negative, then $A_2, A_3, A_4 ...$ are all negative.

$$\sqrt{1-|A|} = 1 - \frac{|A|}{2} - \frac{1}{8}|A_2|^2 \cdot 2^{-4k}$$
$$-\frac{1}{4}|A_2||A_3| \cdot 2^{-5k} - \frac{1}{16}|A_2|^3 \cdot 2^{-6k}$$

- Two's complement of A is taken after (2) to avoid overflow in (3) when using negative numbers directly. This automatically gives the values of $|A_2|, |A_3|$ etc
- In both positive and negative cases, let $B = \sqrt{1+A}$

Dealing with Denominator

- Denominator $\sqrt{\hat{R}}$ is a $(k+1)$-bit number whose square root can be computed using a small LUT as $(k+1)$ is much smaller than n.
- However, as we do not want to use dividers which increase complexity, we use a LUT to calculate $\frac{1}{\sqrt{\hat{R}}}$ (rounded down), storing n-bit result M .
- These LUTs are generated using Python code.
- Final Output $\sqrt{Y} = \frac{B}{\sqrt{\hat{R}}} = B \cdot M$.
- If we use a minimal-sized multiplier, we need to add 0.5×2^{-4k} to final output.

Limitations and Errors

- Error $< 2.39 \times 2^{-4k}$
- One bit of error may be present at the least significant bit of mantissa. Happens sometimes when exponent if odd.
- Denormal Numbers ($< 2^{-126}$), where hidden bit is zero, are not supported by this paper, as the fixed point module support inputs in $[1, 2)$ only.